Config Converter Documentation

Release 1.1

Kent Coble

Jan 28, 2020

Table of Contents

Eliminating mistakes by making a developer responsible to fix them for you!

CHAPTER 1

Introduction

When you've had to complete as many equipment upgrades as we have, the process of having to generate cutsheets and new configuration files is tedious and, well, kinda boring. "So let's automate it instead," someone said. Lo and behold, we delivered **LIKE A BOSS**.

The Config Generator is designed to take in existing configuration files, jack-to-port documentation, current device model and target model in order to speed up the process of pre-configuring equipment for Next-Gen project deployment. Hopefully we will get around to creating a Cisco-to-Dell and vice-versa extension, but we'll that to Kent - because he slaves away his free time to work for some strange reason.

Contents:

1.1 Getting started with the awesome configuration generator

1.1.1 Installation and Preparation

Before you begin, a few packages are necessary to download and install. You can do this via `pip` for easy installation. You can simply type `pip install -r requirements.txt` (or `pip install -r dev-requirements.txt` if you plan on contributing).

Dependencies

```
natsort==5.0.1
ciscoconfparse==1.2.40
colorama==0.3.7
ipaddr==2.1.11
dnspython==1.14.0
openpyxl==2.3.5
et-xmlfile==1.0.1
jdcal==1.2
```

Developer Dependencies

```
alabaster==0.7.8
Babel==2.3.4
ciscoconfparse==1.2.40
colorama==0.3.7
dnspython==1.14.0
docutils==0.12
et-xmlfile==1.0.1
imagesize==0.7.1
ipaddr==2.1.11
jdcal==1.2
Jinja2==2.8
MarkupSafe==0.23
natsort==5.0.1
openpyxl==2.3.5
Pygments==2.1.3
pytz==2016.4
six==1.10.0
snowballstemmer==1.2.1
Sphinx==1.4.4
sphinx-rtd-theme==0.1.9
```

1.1.2 Directory Structure

In order to run the script, several folders need to be created:

```
configconverter
|\_ configs
|\_ cutsheets
\\_ output
 \_ templates
```

1.1.3 Invoking the module from CLI

The module includes an `if __name__ == '__main__':` statement so it can be called from the directory itself. You may copy it directly, but we've included it here (with the `import`) for convenience:

```
from configconverter import SwitchConfigGenerator

if __name__ == "__main__":
    oldconfig, newconfig = get_configs()
    switch_type = get_switch_model()
    hostname = force_user_input("Enter hostname of new switch: ").upper()
    outputfile = input(
        "Enter output file name (default - " + hostname + ".txt): ")
    outputfile = outputfile if outputfile else hostname + ".txt"
    createconfig = ("n" not in input(
        "Generate full config file?[Y|n]: ").lower())

    blades, nojacks, newjacks = migrate_ports(
        oldconfig, newconfig, hostname, switch_type)
    # Add ip dhcp snooping later! Adding it immediately after interfaces
    # causes a bug if trying to use file as startup-config
```

(continues on next page)

(continued from previous page)

```
    vlans = vlan_extract(oldconfig, newconfig, feed_ports_regex[
        switch_models[switch_type]], createconfig)
    setup_feeds(newconfig, switch_type, blades, vlans)
    interfaces_for_review(newconfig, nojacks, newjacks)
    set_voice_vlan(oldconfig)
    access_cleanup(newconfig)
    trunk_cleanup(newconfig)

    # This must be run AFTER trunk_cleanup()
    if not switch_models[switch_type] == "3560":
        remove_mdix_and_dot1q(newconfig)
    if createconfig:
        baseconfig = ".txt"
        if (switch_type == len(switch_models) - 1):
            baseconfig = "baseconfig.txt"
        else:
            baseconfig = switch_models[switch_type] + "base.txt"
        newconfig.prepend_line("!")
        newconfig.prepend_line("hostname " + hostname)
        newconfig.prepend_line("!")
        newconfig.commit()
        extract_management(oldconfig, newconfig)
        add_snooping(newconfig, vlans)
        newconfig.append_line("!")
        with open(template_dir + baseconfig, "r") as b:
            for line in b:
                newconfig.append_line(line.rstrip())
        newconfig.commit()

    file_export(outputfile, newconfig)
```

Note: You are still responsible for including the module directory in the search path

1.2 Order of operation

Since the config generator is far from perfect – unlike myself – there are a few functions that can wreck havoc on your configuration output if you're not careful. (Using the example layout from the script itself works fine.) When you import the module for use in other scripts, be sure to:

1.2.1 `migrate_ports()` before `vlan_extract()`

`vlan_extract()` will also prune any VLANs that are not assigned to edge ports so as to clean out the VLAN database.

1.2.2 `trunk_cleanup()` before `remove_mdix_and_dot1q()`

If ports are configured with `switchport mode trunk` but still contain commands for access ports, all encapsulation configuration will be removed

1.2.3 Call `add_snooping()` last

Adding `ip dhcp snooping` too early will cause the switch to believe that it should be applied to an interface. Since it is not a valid *interface* command, it discards it from the startup-config. It should be invoked last with a least one function called in-between it and `migrate_ports()`, that way a buffer is put between the configuration generated by them.

Note: It should also be called after `migrate_ports()` so that pruned VLANs are not added to the list

Example

Calling too early:

```
interface vlan 300
 ip address 10.23.21.4 255.255.255.128
 no ip route-cache
! The switch will mistakenly apply the next line to VLAN 300 and discard
ip dhcp snooping vlan 24,109,209,309,318,483,509,609,620,651,709,902,985,1902
no ip dhcp snooping information option
ip dhcp snooping
```

1.3 Just need port configuration? I gotchu

If the managed device is setup correctly but ports have had a shift in configuration, you can generate just the list of ports so as to not risk accidentally overriding any management settings. You will still have the option to configure feeding ports too, if desired. The following snippet would be sufficient:

```
from configconverter import SwitchConfigGenerator

if __name__ == "__main__":
    oldconfig, newconfig = get_configs()
    switch_type = get_switch_model()
    hostname = force_user_input("Enter hostname of new switch: ").upper()
    outputfile = input(
        "Enter output file name (default - " + hostname + ".txt): ")
    outputfile = outputfile if outputfile else hostname + ".txt"
    createconfig = ("n" not in input(
        "Generate full config file?[Y|n]: ").lower())

    blades, nojacks, newjacks = migrate_ports(
        oldconfig, newconfig, hostname, switch_type)
    # Add ip dhcp snooping later: adding it immediately after interfaces
    # causes a bug if trying to use file as startup-config
    vlans = vlan_extract(oldconfig, newconfig, feed_ports_regex[
        switch_models[switch_type]], createconfig)
    setup_feeds(newconfig, switch_type, blades, vlans)
    interfaces_for_review(newconfig, nojacks, newjacks)
    set_voice_vlan(oldconfig)
    access_cleanup(newconfig)
    trunk_cleanup(newconfig)
    file_export(outputfile, newconfig)
```

1.4 SwitchConfigGenerator API

SwitchConfigGenerator.**access_cleanup**(*newconfig*)
Remove trunk configuration for all ports set to access mode

Parameters

- **oldconfig** – CiscoConfParse object of existing configuration file
- **newconfig** – CiscoConfParse object, representing the "new" config file

SwitchConfigGenerator.**add_snooping**(*newconfig*, *vlans*)
Add DHCP snooping commands to new configuration file

Parameters

- **newconfig** – CiscoConfParse object, representing the "new" config file
- **vlans** – List of VLANs to add

SwitchConfigGenerator.**add_voice_vlan**(*voicevlan*, *newconfig*)
Add voice VLAN to access ports that do not have it

Todo: Remove *print* statements when *interfaces_for_review* is completed

---

Parameters

- **voicevlan** – a VLAN represented as a string or int
- **newconfig** – CiscoConfParse object representing the "new" configuration file

SwitchConfigGenerator.**condensify_ports**(*ports*)
Turn a collection of ports into a Cisco-formatted range

Todo: Altering format depending on "new" switch model

Parameters **ports** – List of port names

Returns Ports in a Cisco-formatted range

Return type String

SwitchConfigGenerator.**extract_management**(*oldconfig*, *newconfig*)
Extract the management VLAN and add it to new config file

Assuming the target equipment is a layer 2 switch with only one management VLAN, the VLAN config is extracted and the option to retain IP information is provided. *ip tacacs source-interface <VLAN>* is added but only necessary for a 4506; this command will be ignored on all other models.

Parameters

- **oldconfig** – CiscoConfParse object of existing configuration file
- **newconfig** – CiscoConfParse object, representing the "new" config file

SwitchConfigGenerator.**file_export**(*outputfile*, *newconfig*)
Save current configuration to a file

Exports to the directory defined by internal/global var 'output_dir'

Parameters

- **outputfile** – Desired file name
- **newconfig** – CiscoConfParse object, representing the "new" config file

SwitchConfigGenerator.**force_user_input**(*display*, *expect=''*)
Enforce that the user input at least one character

Parameters

- **display** – String to display as the input prompt
- **expect** – Regex string representing the required format of the response before returning to caller. Defaults to an empty string (match any)

Returns User's input

Return type String

SwitchConfigGenerator.**get_configs**()
Set the configuration file to pull data from Prompts user for file name

Returns oldconfig – Existing/source configuration file newconfig – Container for the new device's configuration

Return type CiscoConfParse, CiscoConfParse

SwitchConfigGenerator.**get_switch_model**()
Prompt user to select model from compatible list

Returns The user's input as the internal *switch_models* index

Return type Int

SwitchConfigGenerator.**get_vlan_list**(*oldconfig*, *regex*)
Retrieve all VLANs from the old configuration file and return a list.

(This is intended for future use in cross-platform conversions.)

Parameters

- **oldconfig** – CiscoConfParse object of existing configuration file
- **regex** – Regex string used to determine if port is a feed

Returns All VLANs defined, sorted in ascending order

Return type List

SwitchConfigGenerator.**interfaces_for_review**(*newconfig*, *nojacks*, *newjacks*)
Searches for interfaces on the "new" device with non-standard configs to be reviewed manually.

Searches for statically set PoE, duplex, operating speed, no defined switchport mode

Parameters **newconfig** – CiscoConfParse object representing the "new" configuration file

SwitchConfigGenerator.**is_ip**(*addr*)

SwitchConfigGenerator.**migrate_ports**(*oldconfig*, *newconfig*, *hostname*, *switch_type*)
Map and transfer configuration settings of old ports

Searches for Excel workbooks in the *./cutsheer_dir/* directory. The "cutsheet" files are generated by TurboClerk which are pulled from Netdoc.

As of May 2016, Carlos Bassett has set a standard for worksheet layouts and file names, as follows:

- The spreadsheets must have each tab nammed from the source switch and the first column *MUST* be the port name.
- Any jack associated to this port must be in the row, otherwise configuration will not be transferred.
- The file with the current port-jack mappings must have the building code in the name, along with "as is".
- The file with the future port-jack mappings must have the building code in the name, along with "to be".
- The file with the future port-jack mappings *MUST* begin listing ports in the third row.

Note that this function has the potential to break if corresponding jacks are found from a different existing switch. A future workaround of loading the switch it is found from has been added as a TODO

Todo: Remove *print* statements when *interfaces_for_review* is completed

---

All returned variables are intended for printing out warnings/notices/debugging

Parameters

- **oldconfig** – CiscoConfParse object of existing configuration file
- **newconfig** – CiscoConfParse object, representing the "new" config file
- **hostname** – Name of the new switch
- **switch_type** – the index of switch_models that represents the "new" switch model

Returns _blades_ – Detected blade numbers in the stack _nojacks_ – Port names on the new switch that do not have a jack associated with them _newjacks_ – Port names on the new switch that have jacks associated to them, however they do not exist in any As-Is spreadsheets

Return type (Set, List, List)

SwitchConfigGenerator.**no_files_found**(*directory*)
Allows the user to move files to correct directory or exit early

Parameters **directory** – The folder in which the files should be located

SwitchConfigGenerator.**remove_mdix_and_dot1q**(*newconfig*)
Remove MDIX and dot1q from all interfaces

Note: Should be run after *trunk_cleanup()*

Keyword arguments: newconfig – CiscoConfParse object, representing the "new" config file

SwitchConfigGenerator.**set_voice_vlan**(*oldconfig*)
Select and add a voice VLAN to add to access ports

Parameters **oldconfig** – CiscoConfParse object of existing configuration file

SwitchConfigGenerator.**setup_feeds**(*newconfig*, *switch_type*, *blades*, *vlans*)
Configure feedports

Allows the user to define as many feedports as desired. Checks the validity of the port name as defined by a regex string.

Parameters

- **newconfig** – CiscoConfParse object, representing the "new" config file
- **switch_type** – The representation of the switch model in the form of the switch_models index

- **blades** – A Set of all blade numbers in the stack
- **vlans** – – A List of all VLANs transferred to the new configuration file

SwitchConfigGenerator.**trunk_cleanup**(*newconfig*)
Remove access mode configuration on trunk ports

Removes *access/voice vlan* configs, *spanning-tree portfast*, and *no snmp trap link-status*

Todo: Detect and remove VLANs from VLAN ranges

Parameters **newconfig** – CiscoConfParse object representing the "new" configuration file

SwitchConfigGenerator.**vlan_extract**(*oldconfig*, *newconfig*, *regex*, *genconfig=False*)
Retrieve all VLANs from the old configuration file

Automatically detects if certain VLANs will not be used and will offer to prune them.

Note: For pruning to work, this _must_ be called before *setup_feeds()*

Parameters

- **oldconfig** – CiscoConfParse object of existing configuration file
- **newconfig** – CiscoConfParse object, representing the "new" config file defaults to None
- **regex** – Regex string used to determine if port is a feed
- **genconfig** – A boolean representing if a full config will be generated: If True, all VLANs will be added to the new config file. Defaults to False

Returns All VLANs defined, sorted in ascending order

Return type List

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index